Carnegie Mellon University
Software Engineering Institute

Approaches to Generate Keywords

CERT National Insider Threat Center

June 2021

Cert Division

http://www.sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM21-0551

Table of Contents

1 Introduction

Insider threat analysts require efficacious and reliable methods to generate lists of keywords which can be used to support the detection of a given topic of interest. These keywords may be used to create keyword-based detection policies or may be used by analysts to refer to as a reference guide. Direct keyword detection serves as a generalizable approach, as the complexity, and therefore the ability to take advantage of this method in a variety of tools, is much less than context-based detection. This document presents two approaches for generating lists of keywords by describing the intuition and science behind each approach and by discussing the accompanying software code which performs the automatic keyword extraction. Experimentally, we found that both approaches generated lists of keywords that are reasonably indicative of hate or extremism. We recommend considering the incorporation of these approaches into a keyword development process. However, we also note that a manual review of each of the generated lists of keywords be performed prior to the inclusion of the terms into any automated detection capability. We discuss this note in the Results and Implementation section.

2 Methodology

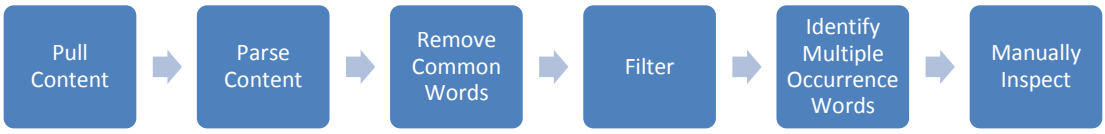
To develop a set of potential keywords and phrases, we followed two separate process, built with a combination of custom Python scripts and open-source tools and libraries. They are described in the next two sections.

2.1 Approach 1: Progressive Filtering

The progressive filtering approach takes heuristically derived rules to extract key terms from hateful and extremist speech. This approach can be configured based upon a given ruleset or provided hateful dataset in order to derive new terms.

2.1.1 Intuition

The concept for the first approach is to find hate or extremist related content, extract uncommon words, filter for known irrelevant terms, and find the list of the remaining words that occur multiple times.



2.1.2 Dataset Requirements

To collect a dataset that reflects the topic of interest (in this case hateful and extremist speech), we looked for user-generated content commonly found on social media platforms and Internet forums. For our experiments, we used the Wayback Machine Downloader[1] to scrape archive.org for HTML content from posts on two known neo-Nazi forums (Fascist Forge and Iron March). This tool downloads the original, latest version of each page on the desired website to a specified directory. For this experiment, we collected approximately 156 megabytes of text content from Iron March on the 08-11-2016 archive of the site just prior to its shutdown. We also collected approximately 30 megabytes of text content from Fascist Forge on the 01-11-2019 archive of the site.

To be able to filter the data down to uncommon words, we needed a dataset that tells how frequently English words are used. For this, we used a Google list of words that had a frequency of

[1] https://github.com/hartator/wayback-machine-downloader

100,000 or more in the Google trillion words corpus, which we extracted from a file downloadable from Kaggle[2]. From the same dataset, we also create lists for bigrams that had a frequency of 10,000 or more and trigrams with a frequency of 1,000 or more. We augmented these with lists of countries, capital cities, and U.S. states.

Finally, we sought to collect a list of other known irrelevant terms. To filter for other known-irrelevant terms, we created a file of nationalities, common first names, ordinal numbers, and common religious words. The information for these were pulled from various lists found doing web searches.

2.1.3 Software Requirements

To run the accompanying software code to automatically extract hateful and extremist keywords using the Progressive Filter approach, the following dependencies must be installed:

- Python 3
- bs4
- lxml
- nltk
- nltk stopwords corpus (on an internet connected system, can be installed by running `import nltk nltk.download('stopwords')`
- nltk punkt corpus (on an internet connected system, can be installed by running `import nltk nltk.download('punkt')`
-

2.1.4 Process

2.1.4.1 Data Retrieval

To get the content for processing, download the site to a dedicated directory. If using a method like `curl` or `wget`, you can use the option to include only .html files. The rest of the process assumes that individual posts or threads are contained in separate index.html files under various subdirectories. If this is not the case, the parseHTML.py script will need to be reconfigured to the provided format.

2.1.4.2 Data Prep

Data prep consists of two components. The first step is to parse the content from the .html files. The parseHTML.py script is used to pull out and do preliminary cleaning of the content and save the posts or threads (the content from each index.html file) to consecutively numbered text files.

The parseHTML.py script should be invoked from the main directory containing the downloaded content. It will walk the directory structure, looking for all index.html files. For each of those files, it will use the BeautifulSoup HTML parser to remove all script, style, meta, link, header,

[2] https://www.kaggle.com/dm4006/google-n-gram

and footer elements. It also removes hexadecimal content. As the output files are named with consecutive integers (e.g., 1.txt, 2.txt) if the process completes, the script will also output a nameMap.txt file that contains a mapping of the numbered files to their location in the directory structure.

The second step required for data prep is to concatenate all of the numbered files into a single text file. Both the concatenated files and individual files are used in data processing[3].

2.1.4.3 Data Processing

Data processing is done with two scripts, keywords.py and coOccurrence.py, run in that order.

The keywords.py script parses the concatenated file into unigrams, bigram, and trigrams and does some initial filtering of the lists. This script uses the following variables for configuration.

- inputFileName: this variable should contain the name of the concatenated content file
- lookupFilePath: this variable should contain the path for the location of the supporting files which come packaged with the code (all should be in the same directory)
    - countries.txt
    - capitals.txt
    - states.txt
    - negAssocWords.txt
    - 100000Plus_freq.txt
    - 1000Plus_2gram.txt
    - 1000Plus_3gram.txt
- removeCountries: this variable should be True if you want to filter out country names
- removeCapitals: this variable should be True if you want to filter out capital cities
- removeStates: this variable should be True if you want to filter out U.S. state names
- removeNum: this variable should be True if you want to filter out numeric only "words"
- removeIgnoreList: this variable should be True if you want to have a custom filter
- ignoreFileName: this variable should have the file name of the custom filter file if removeIgnoreList is True; file contents should be one word per line
- writeUnigram: this variable should be True if you want to write the filtered unigram (single word) list to a file (unigramsOfInterest.txt); if False, will print output to stdout
- writeBigram: this variable should be True if you want to write the filtered bigram (two-word phrases) list to a file (bigramsOfInterest.txt); if False, will print output to stdout
- writeTrigram: this variable should be True if you want to write the filtered trigram (three-word phrases) list to a file (trigramsOfInterest.txt); if False, will print output to stdout
- countNegAssocWords: this variable should be True if you want to print to stout the number of words in the content that have a negative association

The coOccurrence.py script takes a unigramsOfInterest.txt file and the numbered post/thread files and outputs

[3] The individually numbered files are unnecessary if skipping the "identify multiple occurrence words" step. This break-down of steps and scripts allows the process to be run on other data sources, such as electronic books or transcripts. Running the first data processing step against any file would produce a list of uncommon words.

- a list of words from the unigramsOfInterest.txt file that occur in at least a minimum number of files (defined by variable configuration)
- a list of word sets from the unigramsOfInterest.txt file that occur together in at least that minimum number of files.

This script uses the following variables for configuration.

- numFiles: this variable should be set to the number of files of the consecutively numbered content files (e.g., the highest integer value that is a file name)
- countNegAssocWords: this variable should be True if you want to print to stout the number of words in the content that have a negative association.
- levels: this variable should be set to the minimum number of files a word must occur in to be included in the final list; this will result in n!/(k!(n-k)!) set comparisons where n = numFiles and k = levels and higher values for levels drastically increases the number of comparisons; a value of 3 is fine for up-to around 200 files, a value up to 5 is fine for up-to around 45 files; if more files than those limits, consider chunking the file processing
- filterKeywords: this variable should be True if you want to have a custom filter
- filterFileName: this variable should have the file name of the custom filter file if filterKeywords is True; file contents should be one word per line

The keywordsSet.text file contains a list of words that occurs in at least the specified number of files, one word per line. The coOccurrenceList.txt file contains a list of word sets, one word set per line. Each word set is a series of words, pipe-delimited, that all occur in a number of files. For example, say you choose levels = 2 and have the following unigramsOfInterest.txt and four content files:

unigramsOfInterest.txt:

- powers
- people
- rights
- happiness
- unalienable

1.txt: The unanimous Declaration of the thirteen united States of America, When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.

2.txt: We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the pursuit of Happiness.

3.txt: That to secure these rights, Governments are instituted among Men, deriving their just powers from the consent of the governed, --That whenever any Form of Government becomes destructive of these ends, it is the Right of the People to alter or to abolish it, and to institute new Government, laying its foundation on such principles and organizing its powers in such form, as to them shall seem most likely to effect their Safety and Happiness.

4.txt: Prudence, indeed, will dictate that Governments long established should not be changed for light and transient causes; and accordingly all experience hath shewn, that mankind are more disposed to suffer, while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed.

The output of the keywordsSet.txt file would be:

people
powers
rights
happiness

The output in the coOccurrenceList.txt file would be:

people|powers
happiness|rights

2.1.4.4 Finalizing List

The output of the keywordsSet.txt and coOccurrenceList.txt files require manual inspection before implementing as a keyword monitoring list. There are many words that are uncommon, but could be problematic to monitor for, either because they would lead to volumetric overload of results or because there might be legal issues, such as appearance of discrimination. Once the lists are vetted, they can be added to the monitoring tool. Note that results are case *in*sensitive, and do not do word stemming (e.g., removing endings like –s, –ed, or –ing). This should be kept in mind as different monitoring tools may need adjustments to the word lists or rule settings to fit the expectations.

2.2 Approach 2: Supervised Learning

The supervised learning approach applies intuition from hate speech detection literature [1] to derive a set of key keywords that are predictors for the hate speech label. This approach can be configured by varying the training dataset of hate speech and by modifying the processing filters to tokenize the unit of analysis.

2.2.1 Overview

The concept for the second approach is to find hate or extremism related content and benign, control content and use a supervised machine learning approach to learn classification rules and extract the most important features as keywords (in this context, the features are the individual words within labeled records.

2.2.2 Dataset Requirements

For this approach we again based it on user-generated data from social media. We tested using labeled datasets of 28,683 Twitter posts, split between 20, 620 hateful and offensive tweets for the target labels and 8,063 tweets representing normal data derived from a selection of natural disaster related tweets. This approach expects data records in CSV format, with at least two columns: one for the post content and one for the post label. The process uses separate files for the target data and the benign data.

2.2.3 Software Requirements

To run the scripts, the following dependencies must be installed.

- pandas 1.2.0
- scikit-learn 0.24.1
- plotly 4.8.2
- matplotlib 3.3.2
- spacy 3.0.1
- statsmodels 0.12.2

2.2.4 Process

2.2.4.1 Data Retrieval

If using pre-labeled .csv files, for example, from Kaggle, download and if necessary, unzip the data. Otherwise, complete Data Retrieval as described in Approach 1. Repeat for a benign dataset.

2.2.4.2 Data Prep

If not using pre-labeled data, parse the downloaded posts as described in step one of the Data Prep section for Approach 1. Then run labelPosts.py script against the target data files. This file will take the numerically named content files and create a single .csv file containing the content label and the post text. This file has the following variables for configuration:

- numFiles: this variable should be set to the number of files of the consecutively numbered content files (e.g., the highest integer value that is a file name)
- filePath: this variable should be set to the path containing the numerically numbered files that will make up the content of the .csv
- label: this variable should be set to the text value of the label that you want all post entries to be associated with

- header: this variable should be set to the text values for the column headers; the script generates two column so the format is [“col1_header”,”col2_header”]

The output name of the file is <label>.csv.

Repeat running the labelPosts.py script against the benign data files. Be sure to update the label variable.

2.2.4.3 Data Processing

Data processing is done with a Jupyter Notebook. The notebook uses the following variables for configuration.

- targetCSV: this variable contains the file path and name for the file that contains the data with the target label
- benignCSV: this variable contains the file path and name for the file that contains the data with the benign label
- outputFile_unigram: this variable contains the file name for the unigram (single word) keyword output
- outputFile_bigram: this variable contains the file name for the bigram (two-word phrase) keyword output
- outputFile_trigram: this variable contains the file name for the trigram (three-word phrase) keyword output
- outputFile_overall: this variable contains the file name for the keyword output of considering unigram, bigram, and trigrams at the same time
- labelHeaderName_Target: this variable contains the header name for the label column of the target data file
- contentHeaderName_Target: this variable contains the header name for the content column of the target data file
- labelHeaderName_Benign: this variable contains the header name for the label column of the benign data file
- contentHeaderName_Benign: this variable contains the header name for the content column of the benign data file
- n_keywords: this variable is set to the number of potential keyword entries to output
- replaceLabel_Target: this variable is True if the labels in the target data file should be replaced
- labelReplaceList_Target: if replaceLabel_Target is True, this contains a dictionary of values to replace, e.g., {0: 'hate speech', 1: 'offensive language', 2: 'neither'}
- selectLabel_Target: this variable is True if you want to select only a subset of labels from the target data file
- selectLabelList_Target: if selectLabel_Target is True, this contains a list of the subset of labels to select, e.g., ['offensive language','hate speech']
- replaceLabel_Benign: this variable is True if the labels in the benign data file should be replaced
- labelReplaceList_Benign: if replaceLabel_Benign is True, this contains a dictionary of values to replace, e.g., {1: 'real disaster', 0: 'fake disaster'}

- selectLabel_Benign: this variable is True if you want to select only a subset of labels from the benign data file
- selectLabelList_Benign: if selectLabel_Target is True, this contains a list of the subset of labels to select, e.g., ['real disaster','fake disaster']
- targetLabelList: this variable contains the list of labels that corresponds to “target”, e.g., ['offensive language','hate speech']
- BenignLabelList: this variable contains the list of labels that corresponds to “benign”, e.g., ['real disaster','fake disaster']

The script will take the data, if necessary, select the relevant labeled data, and create a classification model. The script then extracts the features—either words (for unigrams) or phrases (for bigrams and trigrams) that are most highly correlated with the target label. The output files contain one keyword or key phrase per line.

2.2.4.4 Finalizing List

The output files require manual inspection to select for the words and phrases relevant to the use case.

3 Results and Implementation

3.1 Results

From approximately 1420 total Iron March forum posts, the progressive filtering method produced 12,786 unigrams, 24,906 bigrams, and 26,108 trigrams. The secondary filtering resulted in 1543 potential keywords that were found in at least 3 different posts. From the 122 Fascist Forge posts, the filtering method produced 1193 unigrams, 1782 bigrams, 1667 trigrams, and 86 Keywords that were found in at least 3 different posts.

The supervised learning approach used a dataset of 28,683 Twitter posts, 72% of which were labeled "bad" and 28% were labeled "benign". This approach extracted 500 unigrams, 500 bigrams, and 500 trigrams.

3.2 Implementation

These approaches assume that user-generated content is grouped into some sort of "post" whether that be social media or online forums, or conversations between individuals such as email or instant messaging threads. Download or obtain a sufficient amount of source data that relates to the topic in question, either via externally collected social media or forum posts, or from an internal data collection tool that captures email or IM conversations such as UAM.

3.3 Limitations

These approaches assume that the source input text is already focused on the sentiment in question. Specifically, the assumption is that all the input text is relevant, and the pipeline does not have a mechanism to distinguish text that does not seem to fit the topic in question. Therefore, it is critical that the input text has undergone some level of preprocessing or selection to ensure that the content is relevant. This can be observed in the example output containing benign words or topics of discussion that generally occur on these types of social forums.

In the absence of a dedicated community of interest or belief in a particular topic, it can be difficult to locate enough source data to compile a meaningful set of keywords or phrases. Moreover, the grammar and vocabulary of the test population (the Internet forum, in this case), may differ greatly from the grammar and vocabulary used in the observed population of an insider threat program.